

Evaluation et sélection de modèles

Alexandre Boucaud (CNRS/APC)

Direction de l'innovation et des relations avec les entreprises

cnrs **formation**
entreprises

Imports

```
In [1]: import matplotlib.pyplot as plt
import numpy as np
import warnings
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

warnings.filterwarnings('ignore')

plt.rcParams['figure.figsize'] = (8.0, 6.0)
plt.rcParams['image.interpolation'] = 'nearest'

np.set_printoptions(precision=2)
```

Evaluation des modèles

Matrice de confusion (confusion matrix)

actual negative	True Negative	False Positive
	False Negative	True Positive
actual positive	False Negative	True Positive
	predicted negative	predicted positive

In [2]:

[illegible]

```
In [3]: from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(max_iter=200).fit(X_train, y_train)
y_pred = lr.predict(X_test)
lr.score(X_test, y_test)
```

```
Out[3]: 0.9440559440559441
```

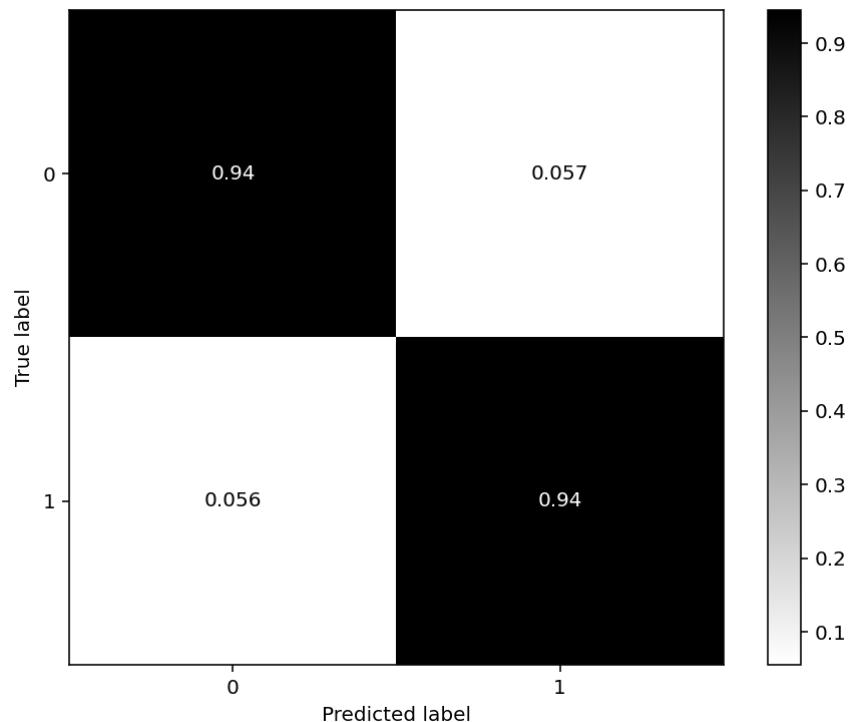
```
In [4]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

print(confusion_matrix(y_test, y_pred))

ConfusionMatrixDisplay.from_predictions(
    y_test, y_pred, cmap='gray_r', normalize='true')
```

```
[[50  3]
 [ 5 85]]
```

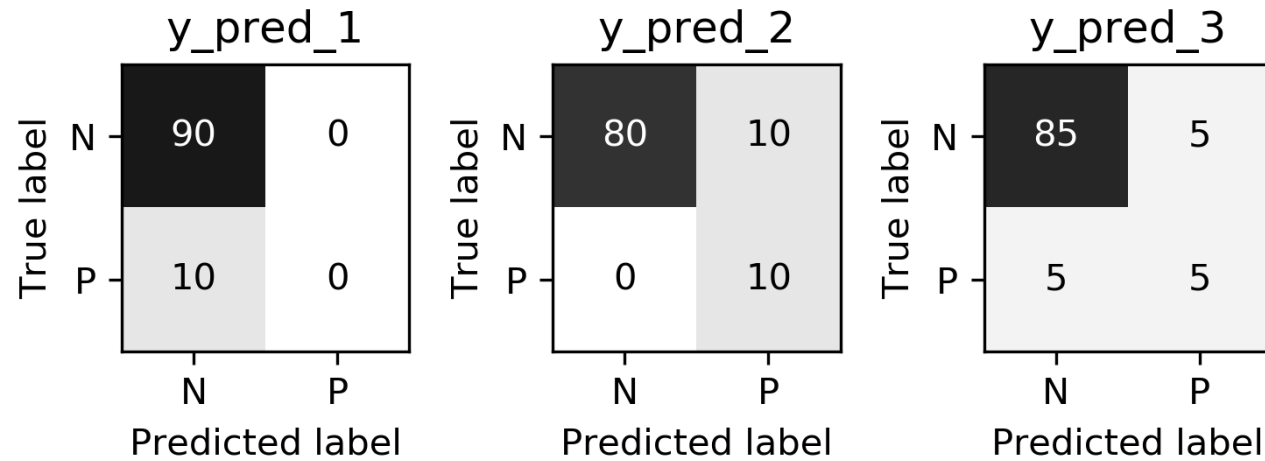
```
Out[4]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x108301c70>
```



Accuracy

$$\text{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$



```
from sklearn.metrics import accuracy_score
```

```
[accuracy_score(y_true, y_pred) for y_pred in (y_pred_1, y_pred_2, y_pred_3)]
```

```
[0.9, 0.9, 0.9]
```

Precision, Recall, f-score and balanced-accuracy

Precision

actual negative	True Negative	False Positive
	False Negative	True Positive
actual positive		
	predicted negative	predicted positive

$$\text{Positive Predicted Value (PPV)} = \frac{TP}{TP+FP}$$

Capacité du modèle à ne pas prédire négatif les exemples positifs

Recall

actual negative	True Negative	False Positive
	False Negative	True Positive
actual positive		
	predicted negative	predicted positive

$$\text{True Positive Rate (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Capacité du modèle à prédire tous les exemples positifs

F1-score

actual negative	True Negative	False Positive
	False Negative	True Positive
actual positive		
	predicted negative	predicted positive

$$\text{F1-score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Moyenne harmonique entre Precision et Recall

Balanced accuracy

actual negative	True Negative	False Positive
	False Negative	True Positive
actual positive		
	predicted negative	predicted positive

$$\text{balanced-accuracy} = \frac{1}{2} \left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP} \right)$$

Meilleure mesure de performance pour les données non-équilibrées (*imbalanced*).

In [5]:

```
from sklearn.metrics import average_precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score

for score in (average_precision_score, recall_score, f1_score, balanced_accuracy_score):
    print(f"{score.__name__}: {score(y_test, y_pred):.2f}")
```

```
average_precision_score: 0.95
recall_score: 0.94
f1_score: 0.96
balanced_accuracy_score: 0.94
```

Rapport de classification

```
In [6]: from sklearn.metrics import classification_report  
report = classification_report(y_pred, y_test)  
print(report)
```

	precision	recall	f1-score	support
0	0.94	0.91	0.93	55
1	0.94	0.97	0.96	88
accuracy			0.94	143
macro avg	0.94	0.94	0.94	143
weighted avg	0.94	0.94	0.94	143

Normalisation de la matrice de confusion

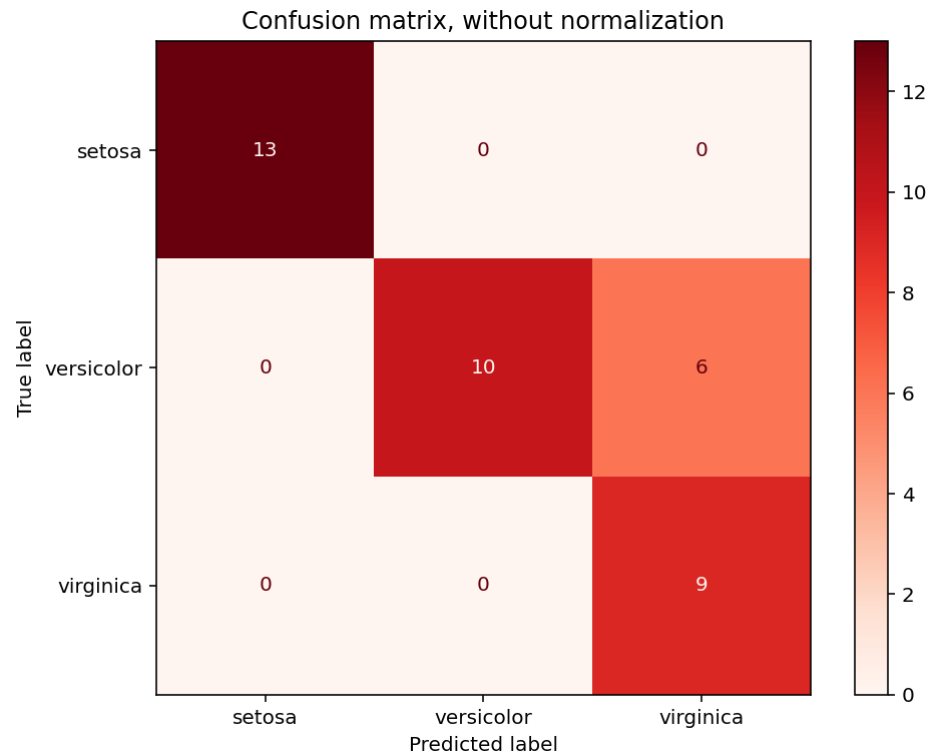
```
In [7]: from sklearn.datasets import load_iris
iris = load_iris()

# Split the data into a training set and a test set
(X_train, X_test,
 y_train, y_test) = train_test_split(iris.data, iris.target, random_state=0)
```

In [8]:

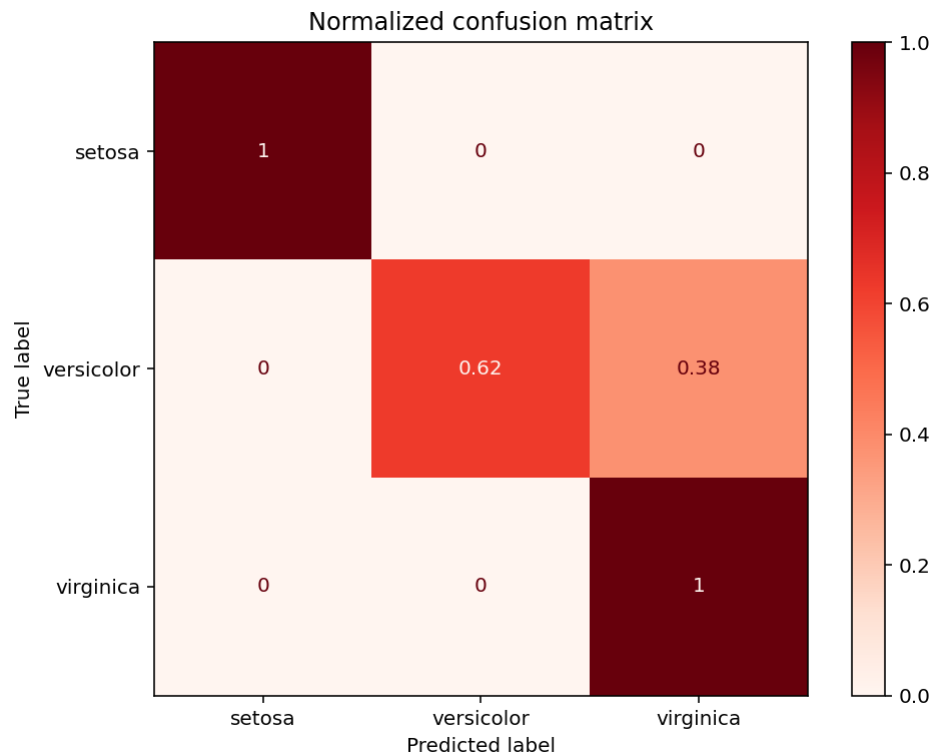
```
from sklearn import svm  
  
classifier = svm.SVC(kernel='linear', C=0.01).fit(X_train, y_train)
```

```
In [9]: disp = ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test,
                                                    display_labels=iris.target_names,
                                                    cmap=plt.cm.Reds,
                                                    normalize=None)
disp.ax_.set_title("Confusion matrix, without normalization")
plt.show()
```



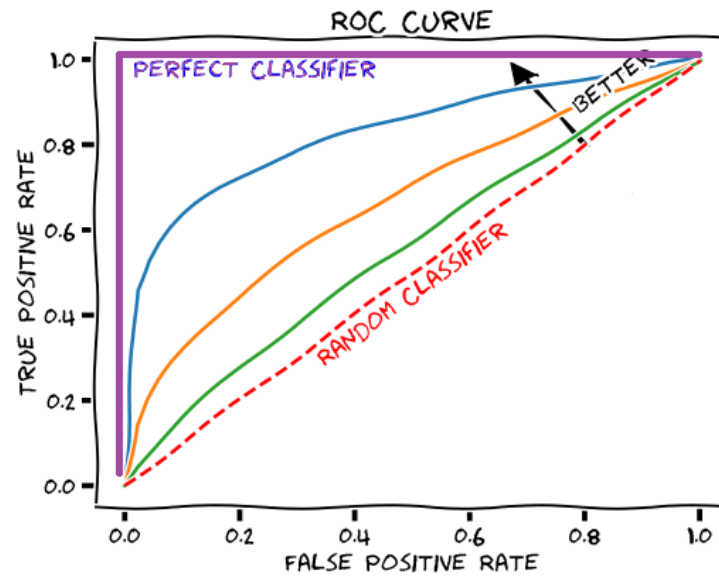
```
In [10]: disp = ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test,
                                                    display_labels=iris.target_names,
                                                    cmap=plt.cm.Red,
                                                    normalize='true')

disp.ax_.set_title("Normalized confusion matrix")
plt.show()
```

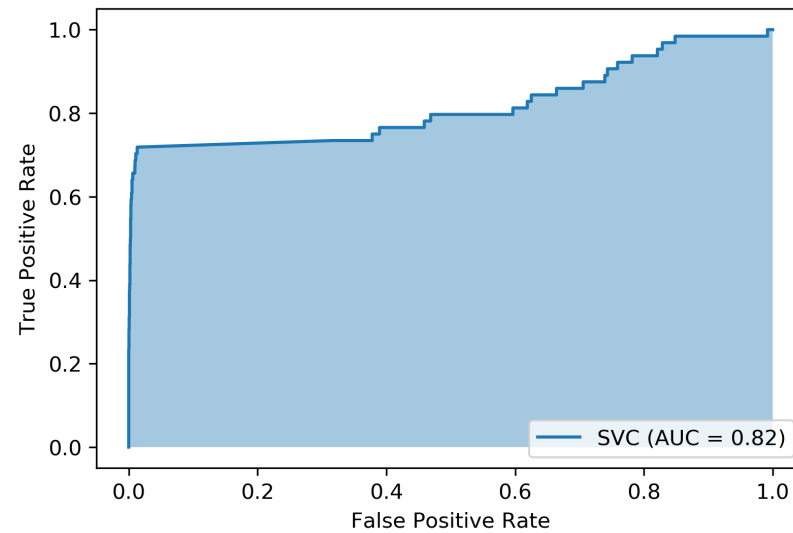


```
In [ ]:
```

Courbes ROC



Area Under Curve (AUC)



```
In [11]: from sklearn.metrics import roc_curve, auc  
         from sklearn.metrics import roc_auc_score
```

In [12]:

[illegible]

In [13]:

```
from sklearn.multiclass import OneVsRestClassifier

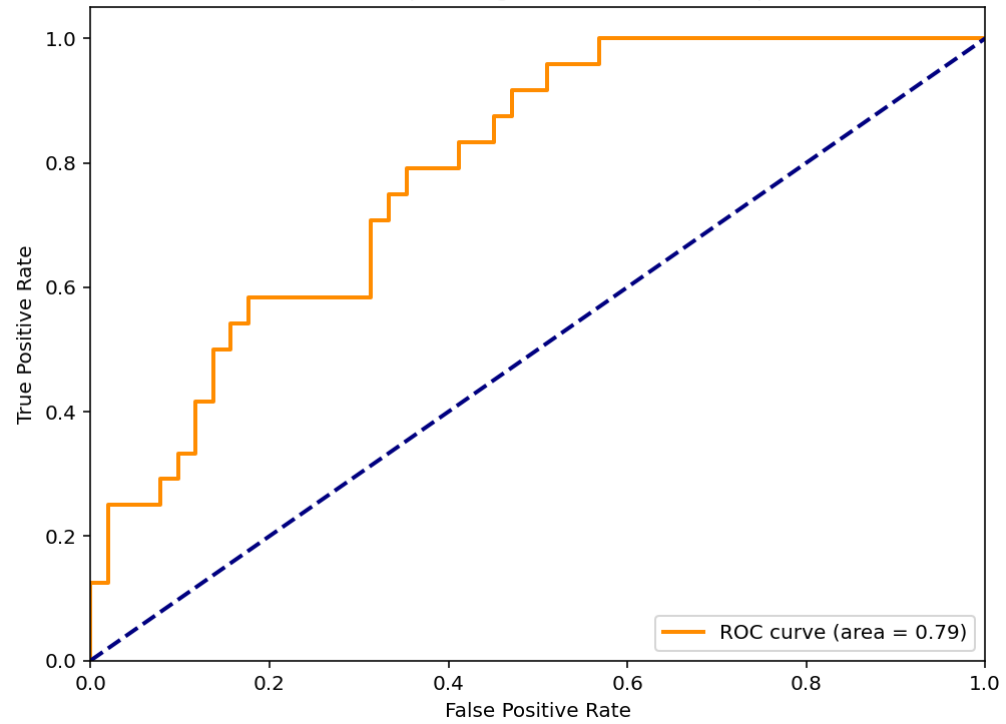
classifier = OneVsRestClassifier(svm.SVC(kernel='linear',
                                          probability=True,
                                          random_state=rng))

y_score = classifier.fit(X_train, y_train).decision_function(X_test)
```

```
In [14]: def plot_roc_curve(class_id):  
    fpr, tpr, _ = roc_curve(y_test[:, class_id], y_score[:, class_id])  
    roc_auc = auc(fpr, tpr)  
  
    plt.figure()  
    plt.plot(fpr, tpr, color='darkorange',  
             lw=2, label='ROC curve (area = %0.2f)' % roc_auc)  
    plt.plot([0, 1], [0, 1], color='navy',  
             lw=2, linestyle='--')  
    plt.xlim([0.0, 1.0])  
    plt.ylim([0.0, 1.05])  
    plt.xlabel('False Positive Rate')  
    plt.ylabel('True Positive Rate')  
    plt.title('Receiver operating characteristic example')  
    plt.legend(loc="lower right")  
    plt.show()
```

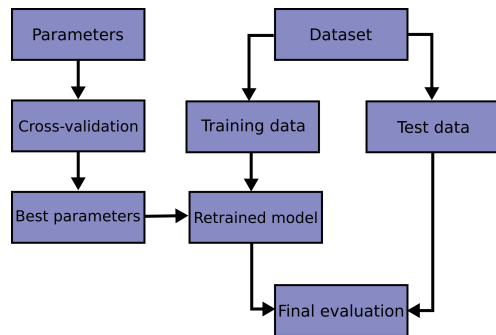
```
In [15]: plot_roc_curve(2)
```

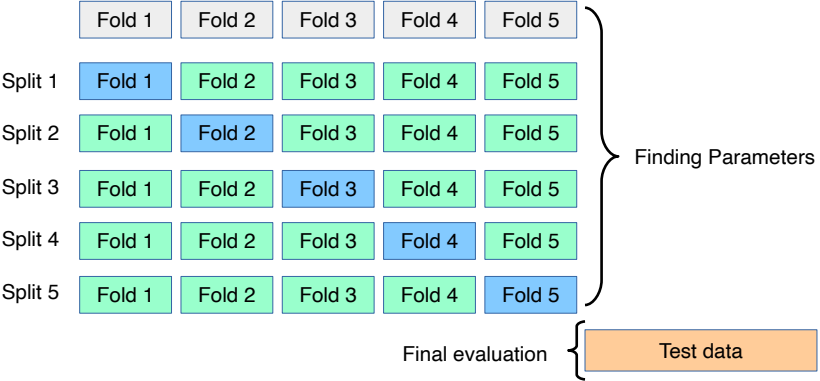
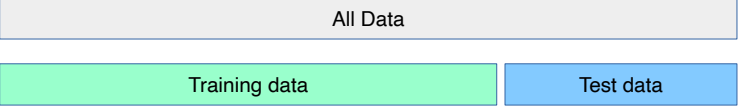
Receiver operating characteristic example



Sélection des modèles

Cross validation score





```
In [16]: from sklearn.model_selection import cross_val_score
```

L'exemple suivant montre comment estimer la performance d'un SVM linéaire sur les données iris, en scindant les données, ajustant un modèle et calculant le score 5 fois consécutives, avec des partitions (*split*) différentes à chaque fois.

```
In [17]: from sklearn.model_selection import cross_val_score
          from sklearn.svm import SVC

          clf = SVC(kernel='linear', C=1)
          scores = cross_val_score(clf, iris.data, iris.target, cv=5)
          scores
```

```
Out[17]: array([0.97, 1.   , 0.97, 0.97, 1.   ])
```


The mean score and the 95% confidence interval of the score estimate are hence given by:

```
In [18]: print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.98 (+/- 0.03)
```

Par défaut, le score calculé à chaque CV est celui associé à l'estimateur. Il est possible de le changer en utilisant `scoring`.

```
In [19]: from sklearn import metrics
scores = cross_val_score(clf, iris.data, iris.target,
                        cv=5, scoring='f1_macro')
scores
```

```
Out[19]: array([0.97, 1.   , 0.97, 0.97, 1.   ])
```

Quand le nombre de `cv` est entier, `cross_val_score` utilise les stratégies `KFold` ou `StratifiedKFold` par défaut. Il est également possible d'utiliser d'autres stratégies en passant directement un itérateur :

```
In [20]: from sklearn.model_selection import ShuffleSplit
n_samples = iris.data.shape[0]
cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
cross_val_score(clf, iris.data, iris.target, cv=cv)
```

```
Out[20]: array([0.98, 0.98, 1.   , 0.96, 1.   ])
```

Itérateurs de cross validation

```
from sklearn.model_selection import KFold

cv = KFold(n_splits=5)

for (idx_train, idx_test) in cv.split(X, y):
    X_train, y_train = X[idx_train], y[idx_train]
    X_test, y_test = X[idx_test], y[idx_test]

    model.fit(X_train, y_train)
    model.score(X_test, y_test)
```

Il existe un nombre important de méthodes pour réaliser cette validation croisée, la plupart sont décrites ici.

https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-iterators

In [21]:

```
def plot_cv_indices(cv, X, y, ax, lw=20):
    """Create a sample plot for indices of a cross-validation object."""
    import numpy as np
    import seaborn as sns
    from matplotlib.pyplot import cm
    from matplotlib.patches import Patch

    splits = list(cv.split(X=X, y=y))
    n_splits = len(splits)

    # Generate the training/testing visualizations for each CV split
    for ii, (train, test) in enumerate(splits):
        # Fill in indices with the training/test groups
        indices = np.zeros(shape=X.shape[0], dtype=np.int32)
        indices[train] = 1

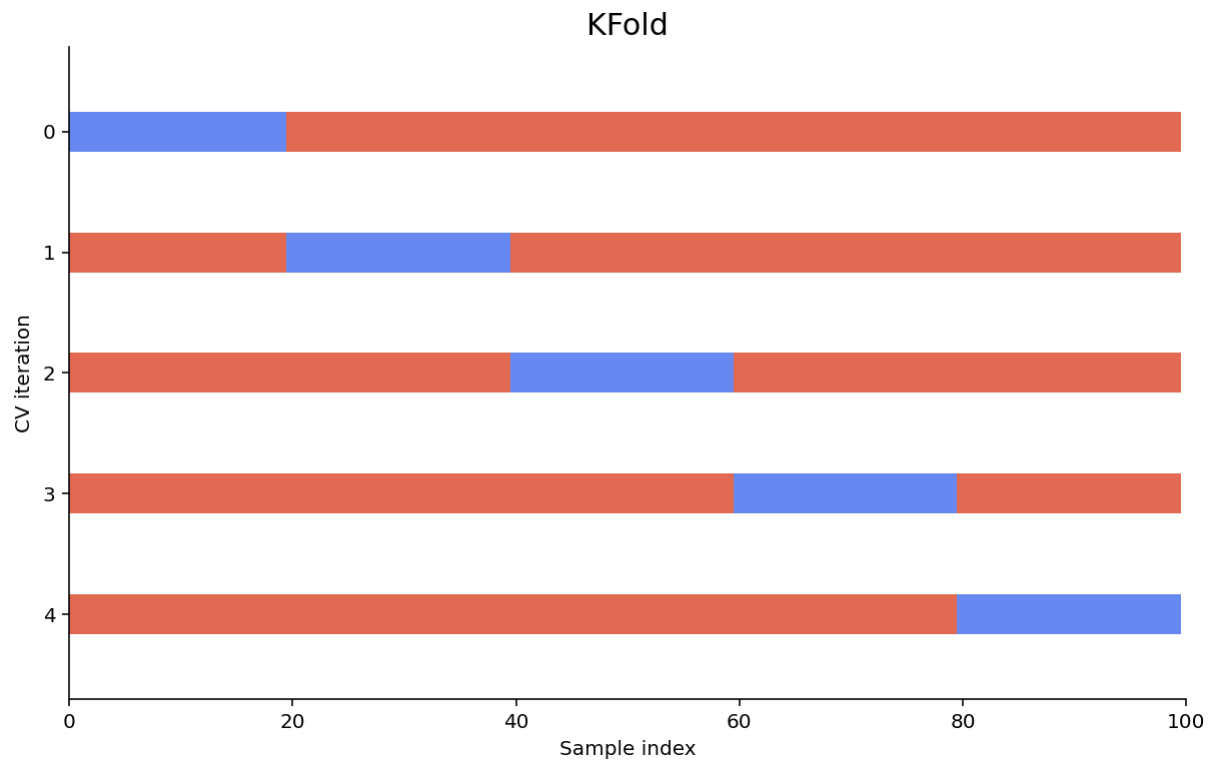
        # Visualize the results
        ax.scatter(range(len(indices)), [ii + .5] * len(indices),
                  c=indices, marker='_', lw=lw, cmap=cm.coolwarm,
                  vmin=-.2, vmax=1.2)

    # Formatting
    yticklabels = list(range(n_splits))
    ax.set(yticks=np.arange(n_splits) + .5,
          yticklabels=yticklabels,
          xlabel='Sample index',
          ylabel="CV iteration",
          ylim=[n_splits + .2, -.2], xlim=[0, 100])
    ax.set_title('{}'.format(type(cv).__name__), fontsize=15)
    sns.despine()
    return ax
```

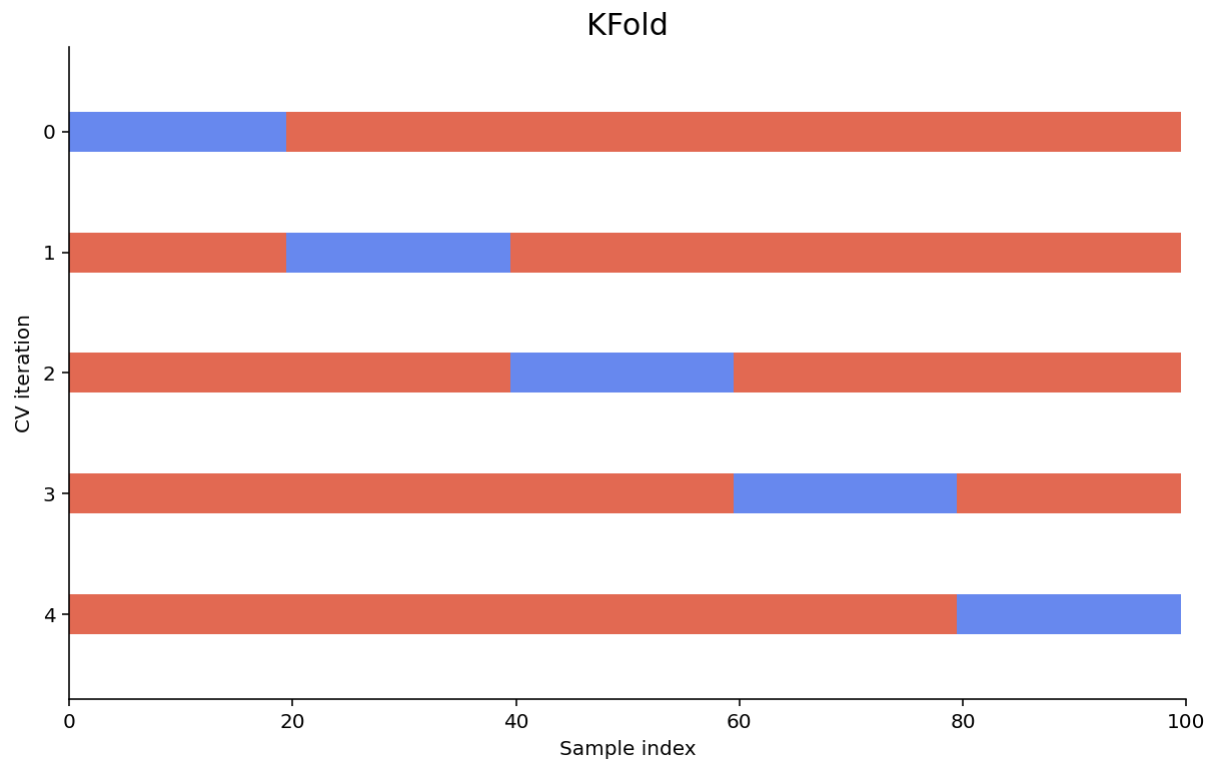
```
In [22]: from sklearn.model_selection import KFold, StratifiedKFold, ShuffleSplit
n_splits = 5

# Some random data points
n_points = 100
X = np.random.randn(n_points, 10)
y = np.zeros(n_points)
y[:20] = 1
```

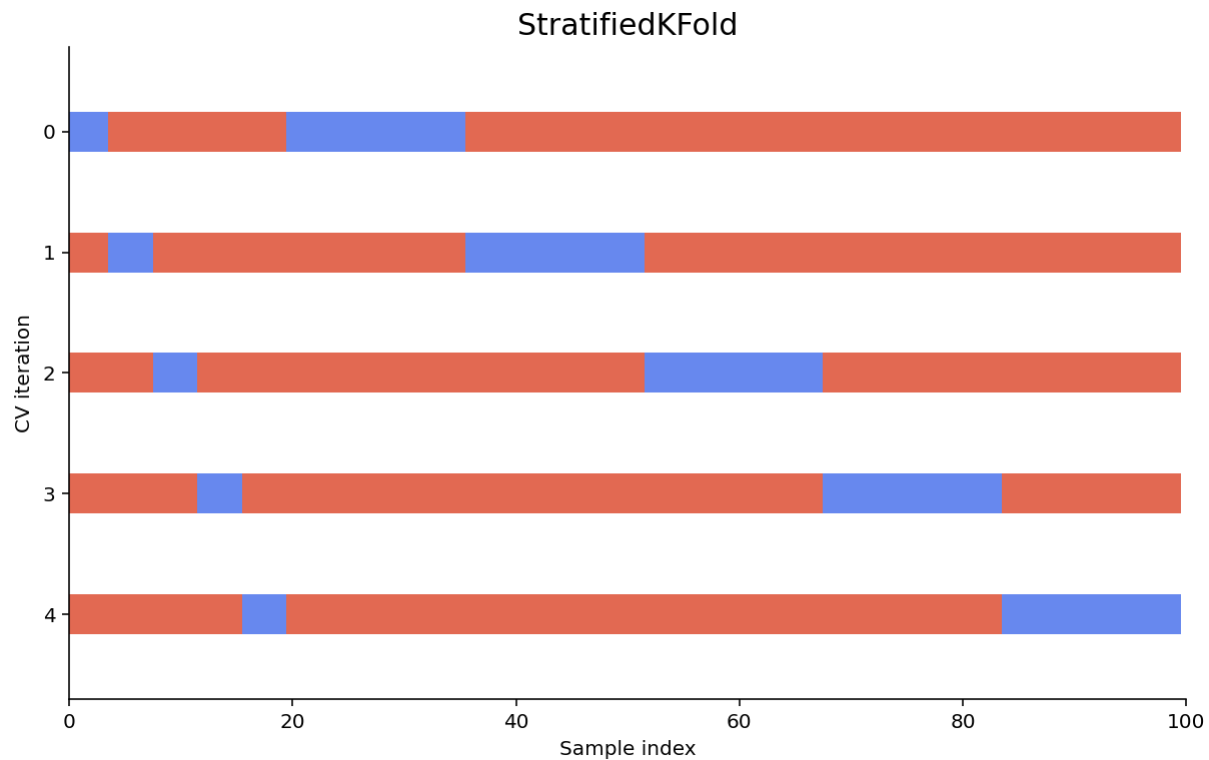
```
In [23]: fig, ax = plt.subplots(figsize=(10, 6))
cv = KFold(n_splits, shuffle=False)
_ = plot_cv_indices(cv, X, y, ax)
```



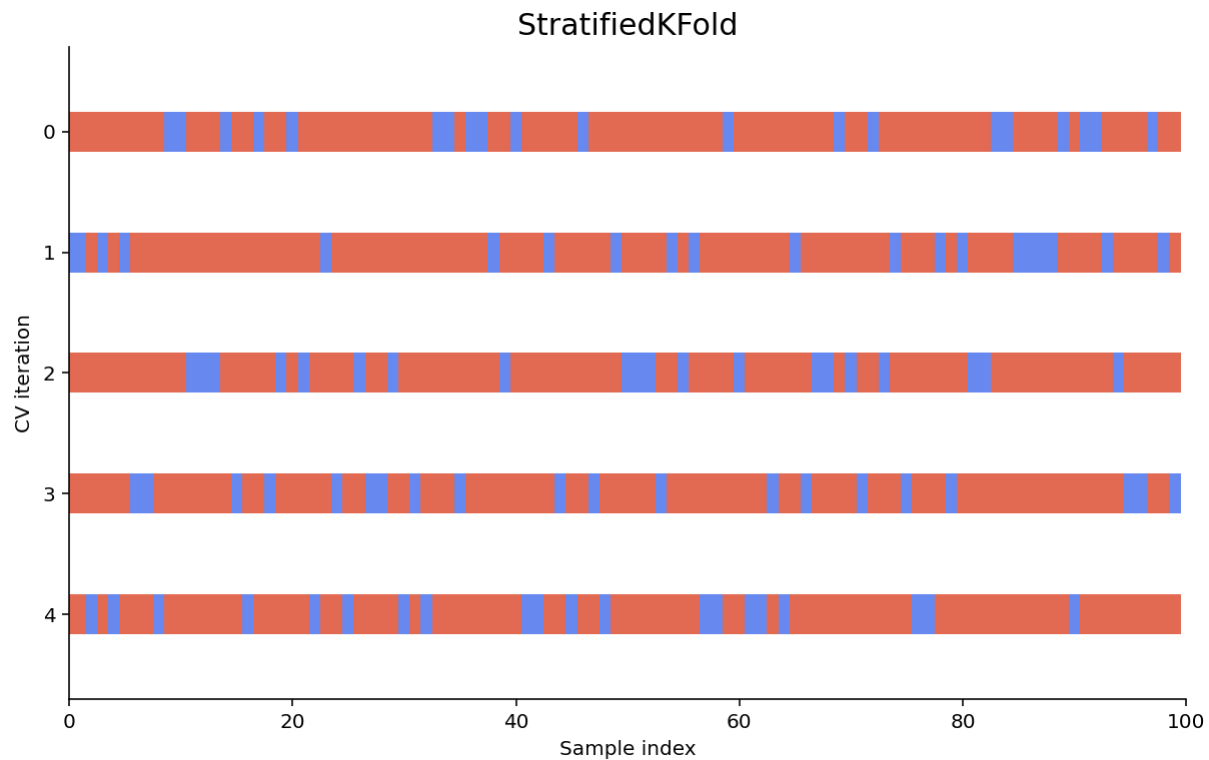

```
In [24]: fig, ax = plt.subplots(figsize=(10, 6))
cv = KFold(n_splits,)
_ = plot_cv_indices(cv, X, y, ax)
```



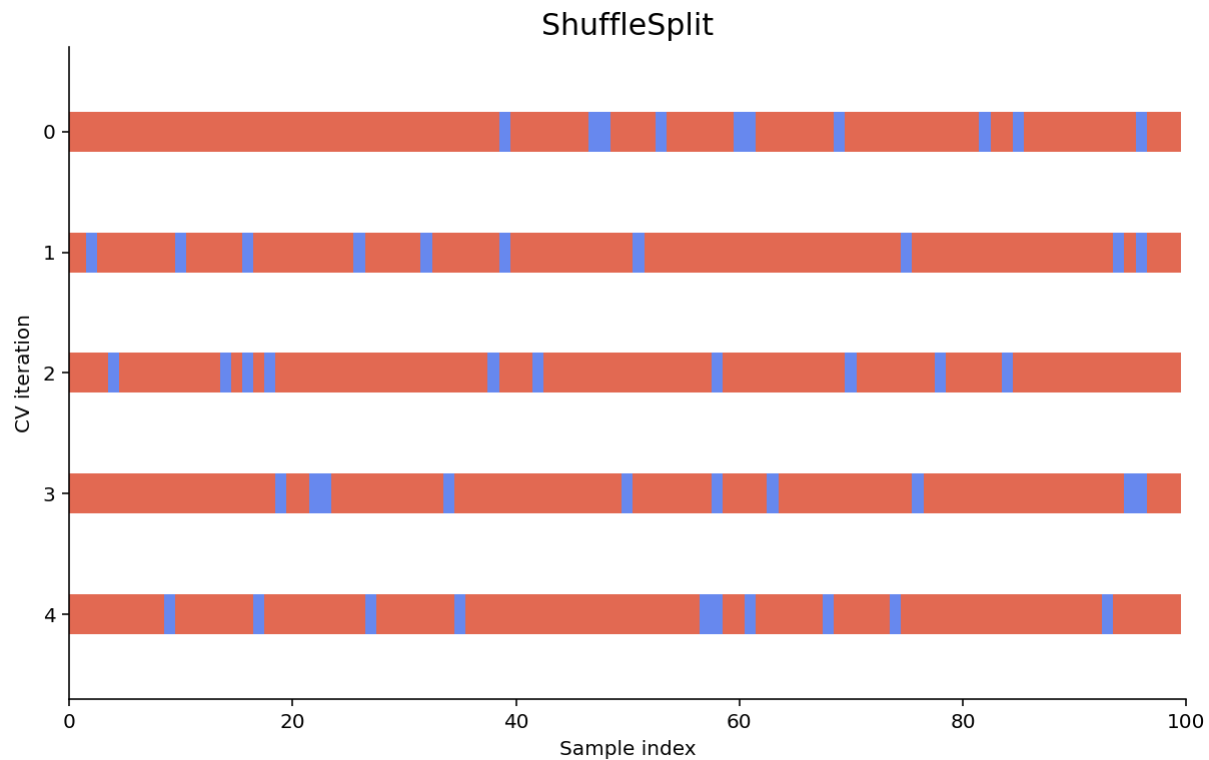
```
In [25]: fig, ax = plt.subplots(figsize=(10, 6))
cv = StratifiedKFold(n_splits, shuffle=False)
_ = plot_cv_indices(cv, X, y, ax)
```



```
In [26]: fig, ax = plt.subplots(figsize=(10, 6))
cv = StratifiedKFold(n_splits, shuffle=True)
_ = plot_cv_indices(cv, X, y, ax)
```



```
In [27]: fig, ax = plt.subplots(figsize=(10, 6))
cv = ShuffleSplit(n_splits)
_ = plot_cv_indices(cv, X, y, ax)
```



Optimisation des hyper-paramètres

In [28]:

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
digits = datasets.load_digits()
```

```
In [29]: # To apply a classifier on this data, we need to flatten the image, to  
# turn the data in a (samples, feature) matrix:  
n_samples = len(digits.images)  
X = digits.images.reshape((n_samples, -1))  
y = digits.target
```

```
In [30]: # Split data into train and test subsets  
(X_train, X_test,  
 y_train, y_test) = train_test_split(X, y, test_size=0.25, shuffle=False)
```

In [31]:

```
from sklearn.svm import LinearSVC

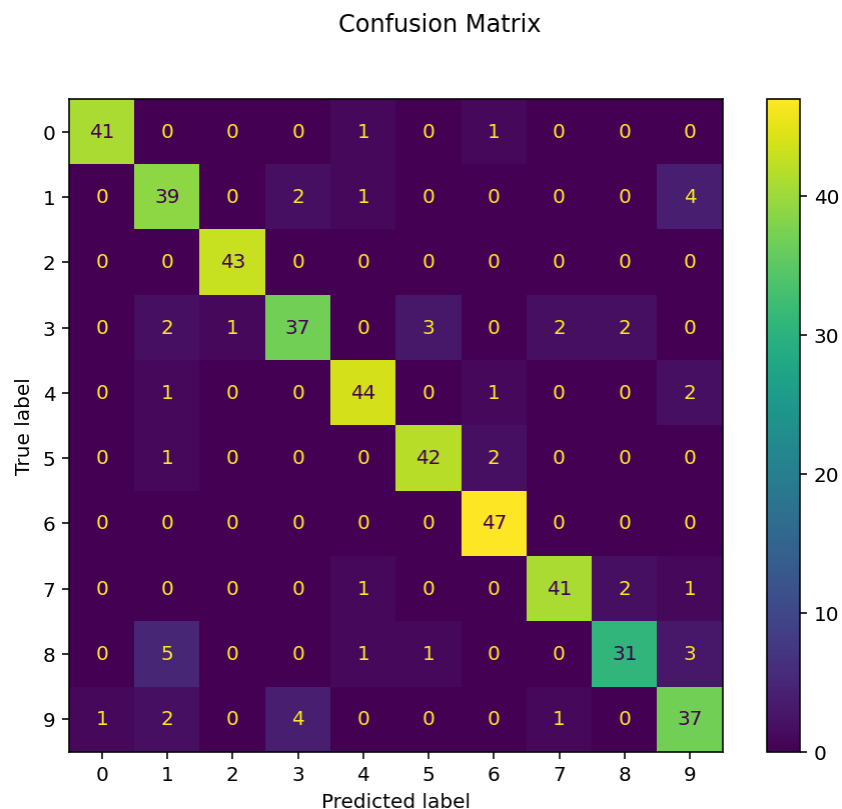
classifier = LinearSVC(random_state=1, max_iter=200).fit(X_train, y_train)
y_pred = classifier.predict(X_test)

print("Accuracy: {}".format(classifier.score(X_test, y_test)))
```

Accuracy: 0.8933333333333333

```
In [32]: disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
disp.figure_.suptitle("Confusion Matrix")
```

```
Out[32]: Text(0.5, 0.98, 'Confusion Matrix')
```




```
In [33]: print("Classification report for classifier %s:\n%s\n"
              % (classifier, metrics.classification_report(y_test, y_pred)))
```

Classification report for classifier LinearSVC(max_iter=200, random_state=1):

	precision	recall	f1-score	support
0	0.98	0.95	0.96	43
1	0.78	0.85	0.81	46
2	0.98	1.00	0.99	43
3	0.86	0.79	0.82	47
4	0.92	0.92	0.92	48
5	0.91	0.93	0.92	45
6	0.92	1.00	0.96	47
7	0.93	0.91	0.92	45
8	0.89	0.76	0.82	41
9	0.79	0.82	0.80	45
accuracy			0.89	450
macro avg	0.89	0.89	0.89	450
weighted avg	0.89	0.89	0.89	450

In [33]:

```
print("Classification report for classifier %s:\n%s\n"  
      % (classifier, metrics.classification_report(y_test, y_pred)))
```

Classification report for classifier LinearSVC(max_iter=200, random_state=1):

	precision	recall	f1-score	support
0	0.98	0.95	0.96	43
1	0.78	0.85	0.81	46
2	0.98	1.00	0.99	43
3	0.86	0.79	0.82	47
4	0.92	0.92	0.92	48
5	0.91	0.93	0.92	45
6	0.92	1.00	0.96	47
7	0.93	0.91	0.92	45
8	0.89	0.76	0.82	41
9	0.79	0.82	0.80	45
accuracy			0.89	450
macro avg	0.89	0.89	0.89	450
weighted avg	0.89	0.89	0.89	450

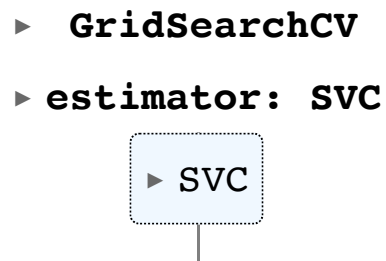
Peut-on obtenir un meilleur score avec le même modèle ?

In [34]:

```
cv = KFold(n_splits=5, shuffle=True)
parameters = {'kernel': ('linear', 'rbf'),
              'C': (0.1, 1, 10),
              'gamma': (0.001, 0.1, 1)}

svc = svm.SVC()
grid = GridSearchCV(svc, parameters, cv=cv, n_jobs=-1)
grid.fit(digits.data, digits.target)
```

Out[34]:



```
In [35]: grid.best_score_
```

```
Out[35]: 0.9899798823893532
```

```
In [36]: grid.best_params_
```

```
Out[36]: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
```

```
In [37]: grid.best_estimator_
```

```
Out[37]: ▼      SVC  
SVC(C=10, gamma=0.001)
```

```
In [38]: grid.cv_results_.keys()
```

```
Out[38]: dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_C', 'param_gamma', 'param_kernel', 'params', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'split3_test_score', 'split4_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score'])
```

```
In [39]: import pandas as pd

cv_results = pd.DataFrame(grid.cv_results_)
cv_results.head(3)
```

Out[39]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	param_kernel	params	split0_test_score
0	0.066583	0.006783	0.018955	0.004502	0.1	0.001	linear	{'C': 0.1, 'gamma': 0.001, 'kernel': 'linear'}	0.75
1	0.253077	0.002548	0.193337	0.005432	0.1	0.001	rbf	{'C': 0.1, 'gamma': 0.001, 'kernel': 'rbf'}	0.75
2	0.055915	0.005485	0.013487	0.002070	0.1	0.100	linear	{'C': 0.1, 'gamma': 0.1, 'kernel': 'linear'}	0.75

In [40]:

```
cv_results_tiny = cv_results[['param_C', 'param_gamma', 'mean_test_score']]
cv_results_tiny.sort_values(by='mean_test_score', ascending=False).head(6)
```

Out[40]:

	param_C	param_gamma	mean_test_score
13	10	0.001	0.989980
7	1	0.001	0.989421
0	0.1	0.001	0.978288
2	0.1	0.100	0.978288
16	10	1	0.978288
4	0.1	1	0.978288

